MOTIVATION

Remote Exploration and Experimentation (REE) Project

Goal: enable new type of scientific investigation by bringing commercial supercomputing technology into space.

This will permit highly autonomous missions with greater flexibility and on-board analysis capability.

Traditional radiation hardening

- Lowers clock speed
- Increases power requirements
- Takes a number of years to complete

Compared to a commodity off-the-shelf component, a radiation-hardened component has a

- Power:performance ratio an order of magnitude lower,
- Cost that is several orders of magnitude higher.

REE Solution

Use COTS hardware and handle resulting faults in software:

- Fault-tolerant system software
- Fault-tolerant routines within applications:
  fault-harden its dominant routines ← Poster focus

Also at Aero-2000: Ferraro et al., "Detailed Radiation Fault Modeling of the REE Testbed Architecture."

SELECTED PREVIOUS WORK

Algorithm-Based Fault Tolerance

- Huang and Abraham[1] introduce algorithm-based fault tolerance, a technique which encodes matrices using checksum matrices. These are then used in order to detect and correct faults in matrix operations. They address matrix operations performed using processor arrays with regard to detecting errors generated by a faulty processor within the array.
- Jou and Abraham[2] present an ABFT error detection scheme for FFT networks. The method employs an encoding and decoding scheme to detect single errors.

Result Checking

- Blum and Wasserman[3] suggest result-checking as a way of enforcing hardware/software reliability. Result-checking relies on developing tests which can confirm the validity of operation output. Their work focuses on computation errors inherent in the system, rather than on environmentally induced faults.

ABFT and Result Checking

For purposes of detecting errors in operations that are expressed in linear form, ABFT and result checking are essentially the same.

[1] K.-H. Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers*, 33(6):518-528, 1984.

[2] J.-Y. Jou and J. A. Abraham. Fault-tolerant FFT networks. *IEEE Transactions on Computers*, 37(5):548-561, 1988.

[3] M. Blum and H.. Wasserman. Reflections on the Pentium division bug. *IEEE Transactions on Computers*, 45(4):385-393, 1996.

FAULT DETECTION

Checksum Methods

Are used to validate results returned by a numerical subroutine.

General Considerations

The *postcondition*: A necessary (but possibly not sufficient) relationship between the subroutine inputs and computed outputs.

Checking the postcondition provides a powerful sanity check on the proper functioning of the subroutine.

Postconditions considered here involve comparing two linear maps, which are known in factorized form:

$$L_1 L_2 \cdots L_p \stackrel{?}{=} R_1 R_2 \cdots R_q \qquad .$$

A typical fault to data fans out across matrix outputs, but a single probe with vector w catches most errors:

$$L_1 L_2 \cdots L_p w \stackrel{?}{=} R_1 R_2 \cdots R_q w$$

Design issues:

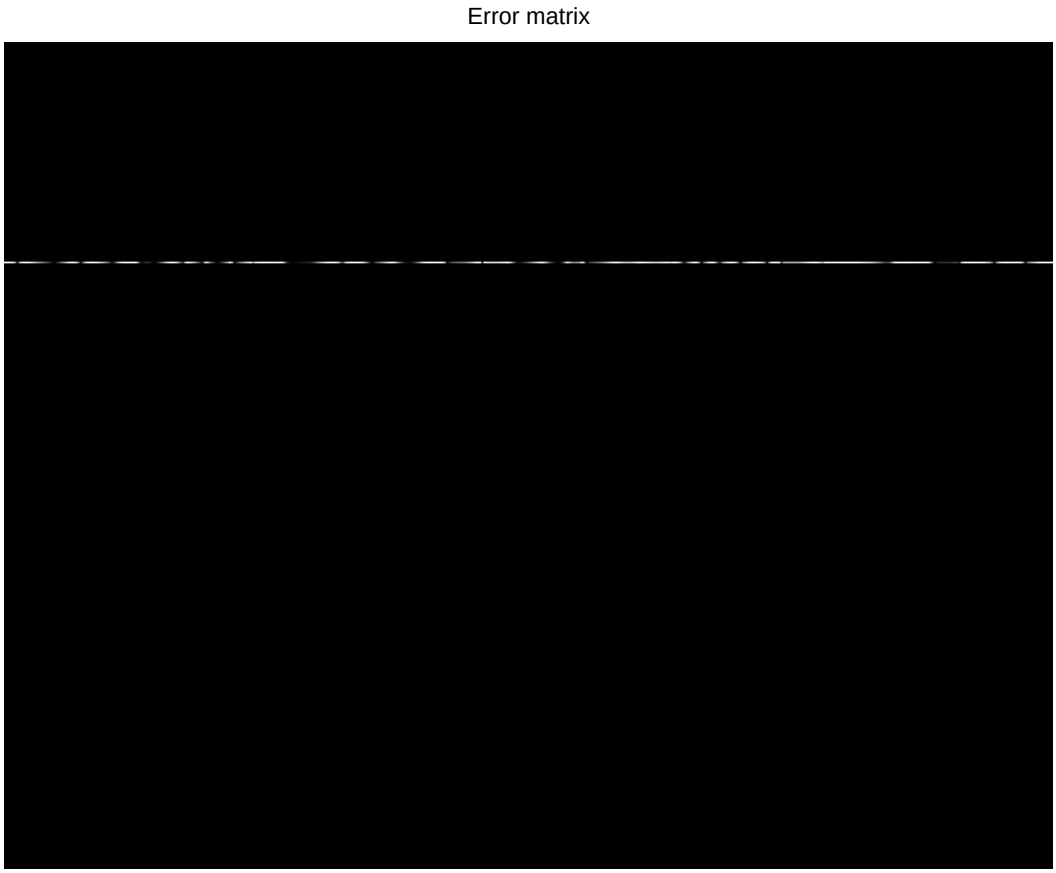
- The choice of the probe vector w.
- The choice of comparison method $\stackrel{?}{=}$.

First is relatively straightforward: elements of w should not vary greatly in magnitude so that results figure equally in the check.

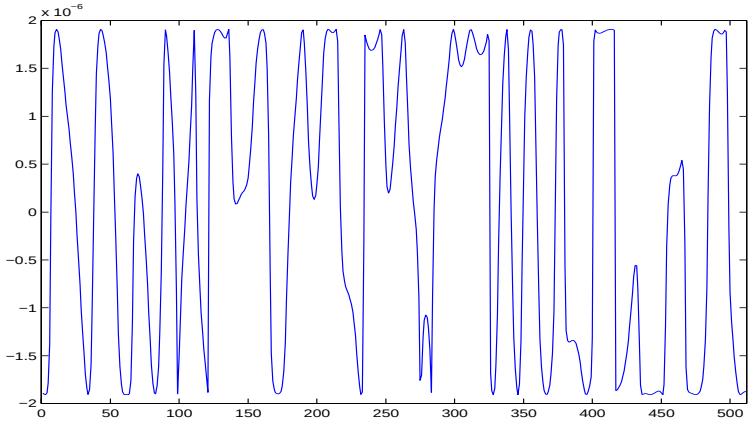
Some algorithms (e.g., FFT) allow tailored choice of w.

FAULTS AFFECTING FFT

Difference matrix caused by a single fault during a 2D FFT in the REE hardware testbed.



Note the fanout during the FFT butterfly calculation in this section through the faulty row.



DEVELOPING COMPARISON TESTS

Operations Considered

- Product: find $\boxed{AB = P}$, given A and B.
- LU decomposition: factor A as $\boxed{A = PLU}$
  P is permutation matrix,
  L is unit lower-triangular,
  U is upper-triangular.
- Singular value decomposition: factor A as $\boxed{A = UDV^{\mathsf{T}}}$
  D is diagonal,
  U and V are orthogonal matrices.
- Matrix inverse: given A, find B such that $\boxed{AB = I}$
- Fast Fourier Transform: given x, find $\boxed{y = Wx}$
  for a fixed transform matrix W.

Postcondition is necessary and sufficient for `mult`, `inv`, `fft`

Comparison Tests

Tests proceed from error matrix $\Delta = RHS - LHS$ and $\delta = \|\Delta w\|$

Error propagation bounds allow input-independent tests.
But different tests have different accuracy/flop tradeoffs

$$
\begin{array}{lrll}
T0: & \delta/\|w\| & \overset{>}{<} \tau\mathbf{u} & \text{(trivial test)} \\
T1: & \delta/(\sigma_1\|w\|) & \overset{>}{<} \tau\mathbf{u} & \text{(ideal test)} \\
T2: & \delta/(\sigma_2\|w\|) & \overset{>}{<} \tau\mathbf{u} & \text{(approximate matrix test)} \\
T3: & \delta/(\lambda\|w\| + \sigma_3) & \overset{>}{<} \tau\mathbf{u} & \text{(approximate vector test)}
\end{array}
$$

$\mathbf{u}$ is the numerical precision of the underlying hardware
threshold τ controls the false-alarm/detection tradeoff

- *Ideal test T1*: derived from error propagation bounds, but may not be computable.
- *Matrix test T2*: approximate, based on computed quantities.
- *Vector test T3*: also approximate, and use of vector norm increases chance of false alarms.

Compile the threshold checks into a table

Algorithm	Δ	σ_1 *(ideal)*	σ_2 *(mat.)*	σ_3 *(vec.)*	Note
`mult`	$\hat{P} - AB$	$\|A\|\|B\|$	$\|\hat{P}\|$	$\|\hat{P}w\|$	—
`lu`	$\hat{P}\hat{L}\hat{U} - A$	$\|A\|$	$\|\hat{P}\hat{L}\hat{U}\|$	$\|Aw\|$	σ_1 easier than σ_2
`svd`	$\hat{U}\hat{D}\hat{V} - A$	$\|A\|$	$\|\hat{U}\hat{D}\hat{V}^{\mathsf{T}}\|$	$\|Aw\|$	σ_1 easier than σ_2
`inv`	$I - A\hat{B}$	$\|A\|\|A^{-1}\|$	$\|A\|\|\hat{B}\|$	$\|A\|\|\hat{B}w\|$	$\|A\hat{B}w\|$ useless
`fft`	$(\hat{y} - Wx)^{\mathsf{T}}$	$\|x\|$	—	—	—
`ifft`	$(\hat{x} - W^{\mathsf{T}}y)^{\mathsf{T}}$	$\|y\|$	—	—	—

Hatted quantities are returned by the algorithm and may be faulty

Any standard vector norm may be used; infinity-norm is easy to calculate

FAULT-DETECTION PERFORMANCE

Simulation Setup I

- Matlab simulation
- A population of random matrices is used as the input.
  Inputs have condition from 2^1 to 2^{20} and are randomly scaled
- Faults are injected in half these computations by first choosing a matrix to affect, and then flipping exactly one bit of its 64-bit representation.
- Operations tested: `mult`, `lu`, `lu`, `svd`.

Simulation Setup II

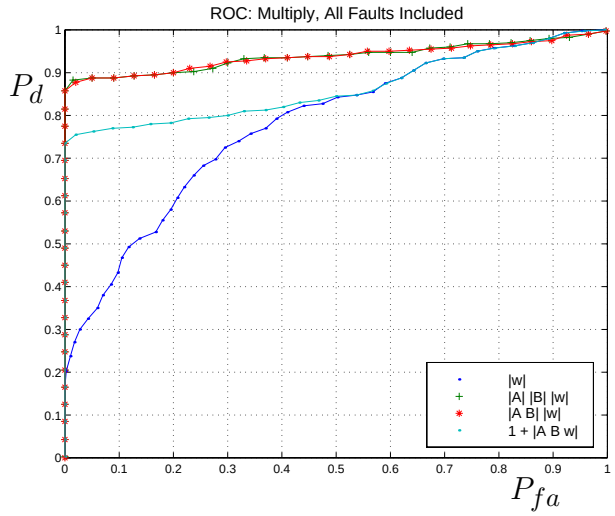
- C program running on the REE interim testbed, a parallel system consisting of 9 processors running the Lynx OS.
- A population of uniform random matrices is used as the input.
- Each operation is done twice — faults are injected during the second computation and the results are compared.
  Zero or more faults may be injected during the faulty run.
- Operation tested: `fft`.

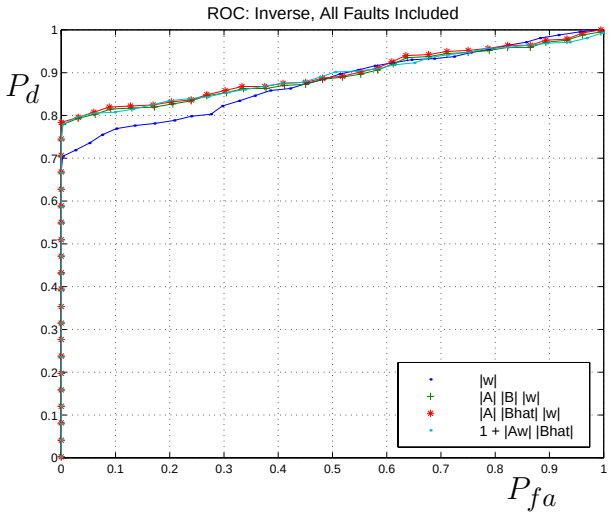
ROC Curves

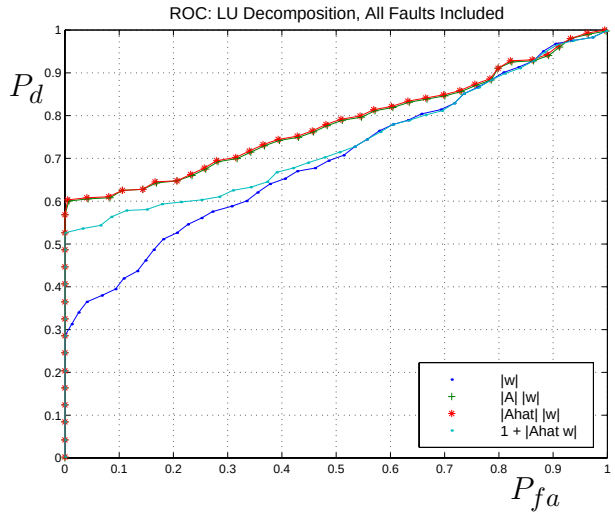
Characteristics of a given scheme are concisely represented using the standard receiver operating characteristic (ROC) curve.
For a given tolerance, some proportion of False Alarms (numerical errors tagged as data faults) and Detections (data faults correctly identified) will be observed.
The ROC plots these two proportions parametrically as tolerance τ is varied.

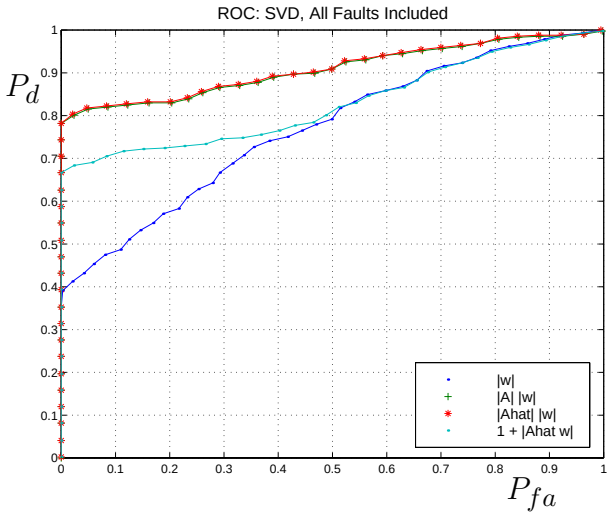
RESULTS: SIMULATION I

ROC Including All Faults









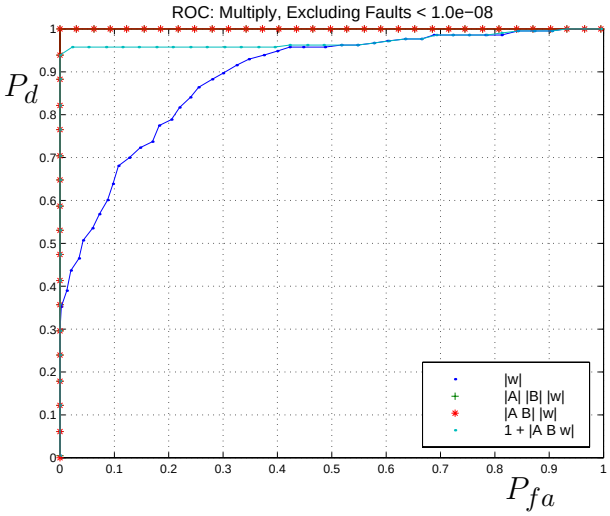
The knees of these curves are rather low: many virtually undetectable faults occur in the low-order bits of the mantissa.

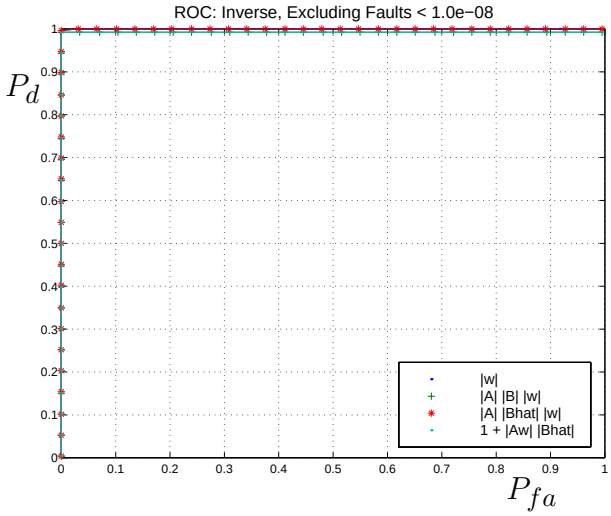
A heavy price is paid for using suboptimal tests $T0$ and $T3$.

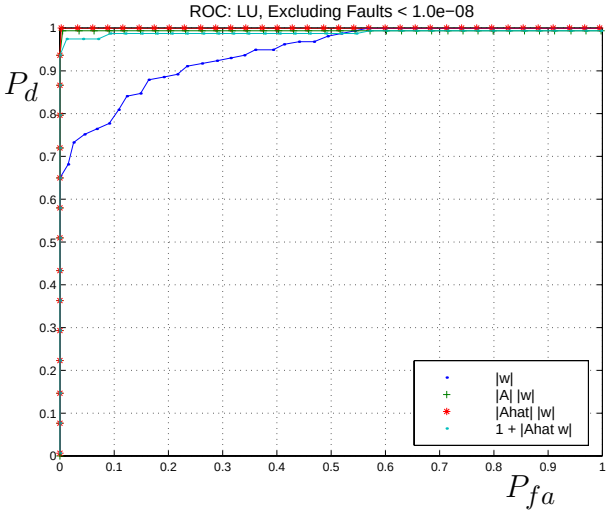
$T1$ and $T2$ show the same performance.

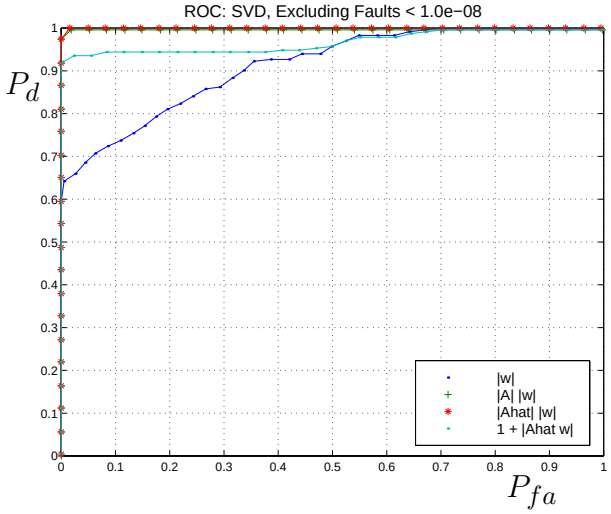
RESULTS: SIMULATION I

ROC Including Significant Faults Only







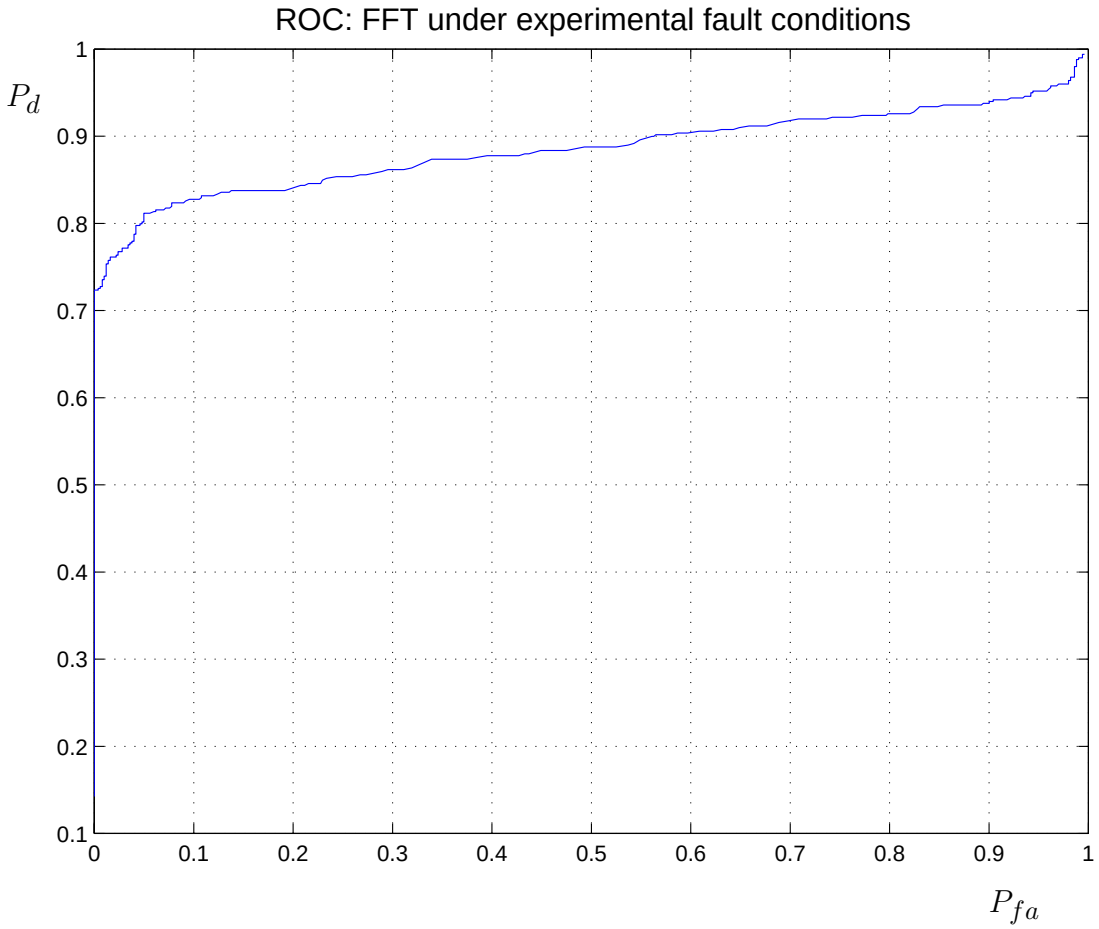


We now exclude faults of relative size less than 10^{-8}.

These curves show the usable performance of the checksum methods.

Only $T1$ and $T2$ may be recommended; they perform well here.

RESULTS: SIMULATION II

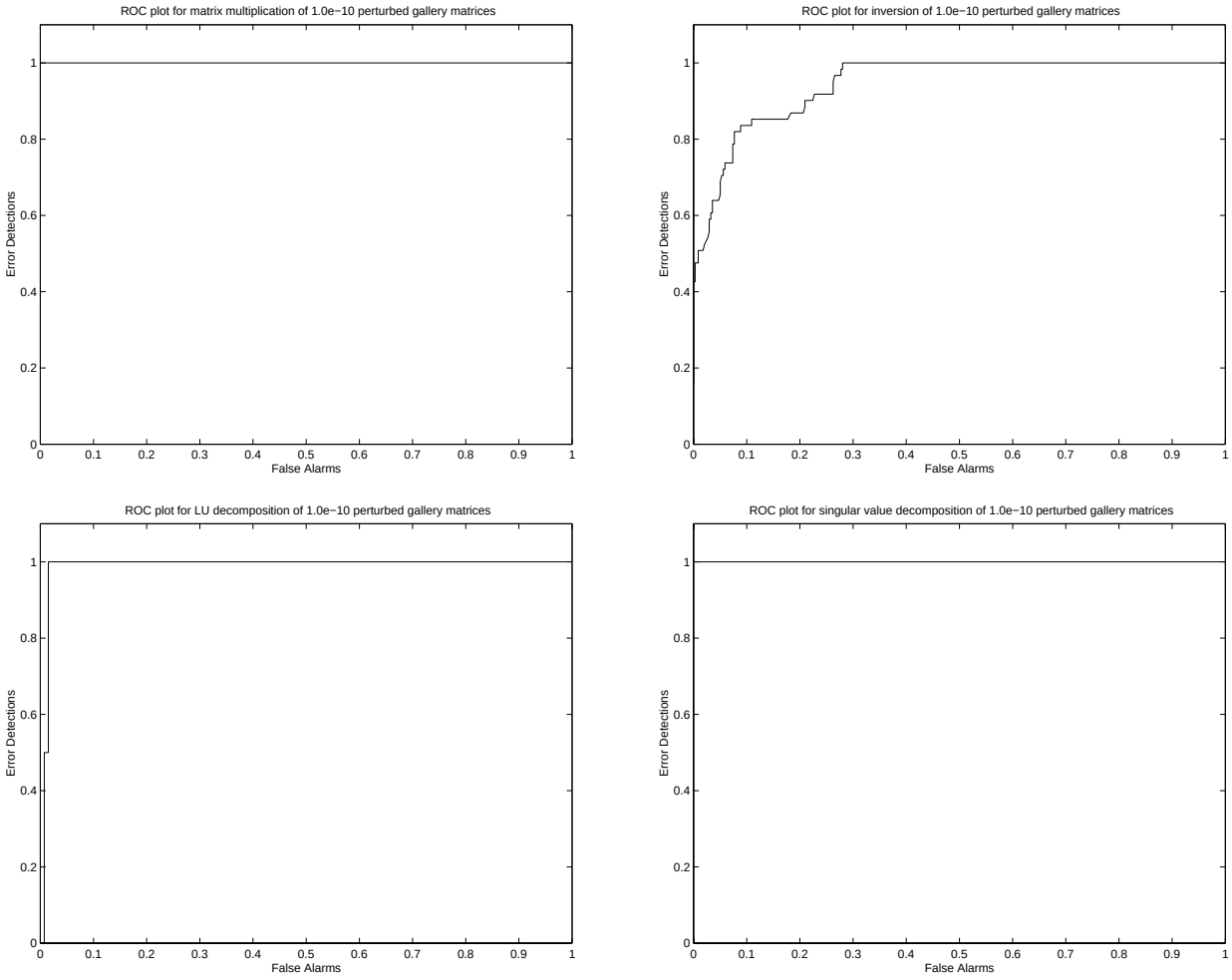


ROC for uniform random matrices, including all faults.

This plot cannot be compared to those in first simulated results: in those examples the injection of a fault is a known event, whereas under this testbed fault environment "real" faults must be determined by a threshold measurement of the output difference.

TESTING THE IMPLEMENTATION

ROC: Testing Parallel Operation



Parallel implementation checked by Matlab

In contrast to the results just reviewed, these curves were generated by checking ScaLAPACK computations with Matlab. We declare that an error has occurred when the two full results differ by more than a fixed tolerance. Errors are detected by the ScaLAPACK implementation.

CONCLUSIONS AND FUTURE WORK

Conclusions

- Theoretical results bounding expected roundoff in computations can provide several types of input-independent threshold tests for checksum differences.
- Readily computable tests are easy to define.
- Under simulated fault injection, the tests for `mult`, `lu`, `svd`, and `inv` behave in good agreement with theory.
- Under more realistic fault injection conditions, a test for `fft` is in good general agreement with theory.
- Tests of the numerical characteristics of our parallel implementation indicate excellent agreement with another numerical package.
- The ABFT `fft` routines have been integrated with an image texture analysis and segmentation application; preliminary tests indicate that the checksum scheme effectively protects the Fourier transform operations within the application.

Future Work

- Testing of all operations under more realistic fault conditions.
- Test routines under within a more extensive software framework, such that programs which have crashed or hung are restarted automatically.
- Integration of the ABFT routines with several science applications, and extensive testing thereof.
- Investigation of the feasibility of using ABFT checksum schemes for low-level routines (addition, multiplication, etc.).
- Hardening of other routines such as sorting, order statistics, and numerical integration.